

The background of the slide is a dark blue gradient. It features a complex geometric pattern of concentric circles and intersecting lines. There are three main sets of concentric circles, each centered at different points on the slide. These circles are intersected by several straight lines, some of which are solid and others dashed, creating a web-like structure across the entire background.

Communication Support for Global Address Space Programming Models

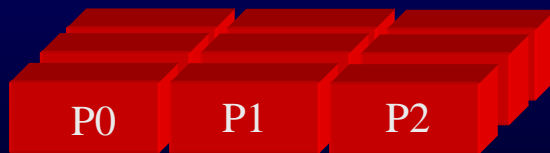
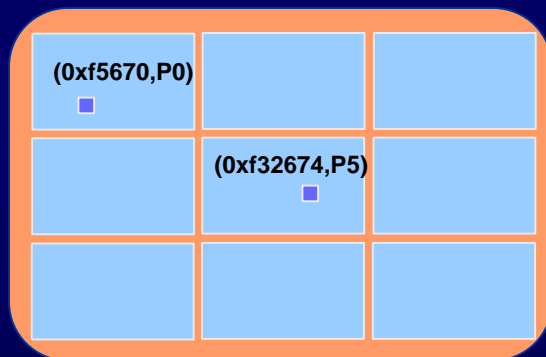
Jarek Nieplocha
Pacific Northwest National Laboratory

Outline

- Background
 - Underlying Concepts
 - DoE MICS Project on Programming Models for Scalable Parallel Computing
- Specific programming models
- Run-time System efforts
- Relation to Blue Gene/L system

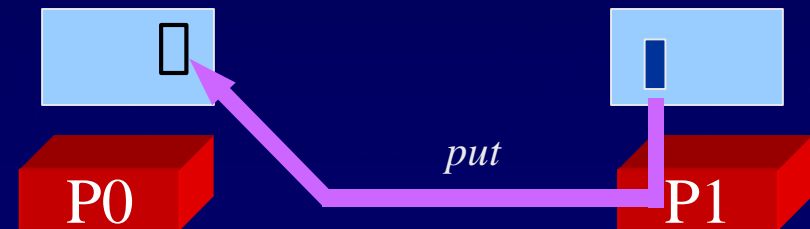
Global Address Space and Remote Memory Access

*collection of address spaces
of processes in a parallel job*
global address: (address, PID)



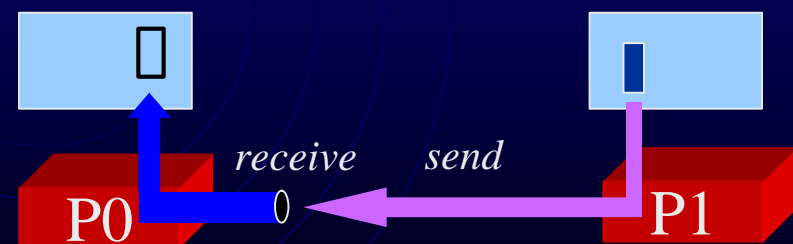
Examples: Cray T3E, Fujitsu VPP5000

associated communication paradigm



Remote Memory Access

rather than



Point-to-point Message Passing



GAS Models in PModels Project

www.pmodels.org

- Global Address Space is a concept shared among several models in the PModels project
 - Language Based Programming Models
 - Co-Array Fortran (Rice, U. Minnesota)
 - UPC (LBNL, Berkeley)
 - Titanium (Berkeley)
 - Library Based Programming Models
 - MPI-2 1-sided (Argonne)
 - SHMEM (Ames Lab)
 - Global Arrays (PNNL)

CoArray Fortran

- Relatively simple extensions (comparing to HPF) to Fortran 90
- SPMD process images
 - number of images fixed during execution
 - images operate asynchronously
- Both private and shared data
 - allows programmer manage data locality explicitly
- `real a(20,20)` private: a 20x20 array in each image
- `real a(20,20) [*]` shared: a 20x20 array in each image
- Simple one-sided shared memory communication
 - `x(:,j:j+2) = a(r,:) [p:p+2]` copy 3 rows from p:p+2 into 3 local columns
- Flexible synchronization
 - `sync_team(team [,wait])`
 - `team` a vector of process ids to synchronize with
 - `wait` a vector of processes to wait for (a subset of team)
- Pointers and (possibly asymmetric) dynamic allocation
- Parallel I/O (Panda/HDF efforts at UIUC & NCSA)

UPC

- Shared array elements are spread across the threads

shared int x[THREADS] one element per thread

shared int y[3][THREADS] 3 elements per thread

shared int z[3*THREADS] 3 elements per thread, cyclic

Assume THREADS=4, elements with affinity to processor 0 are marked

x 

y 

z 

blocked

cyclic

This is really
a 2D array

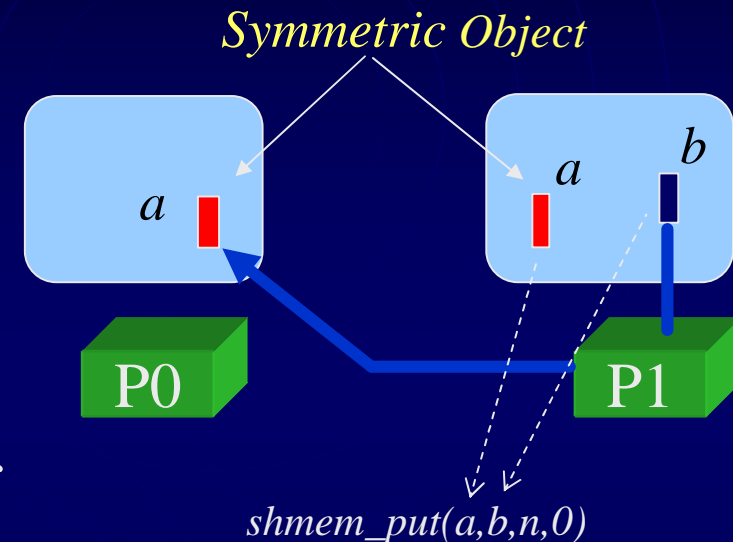
- Pointers may point to shared or private variables

shared int *sp; pointer to an integer residing in the shared memory

- Locks
- Strict and relaxed memory consistency models

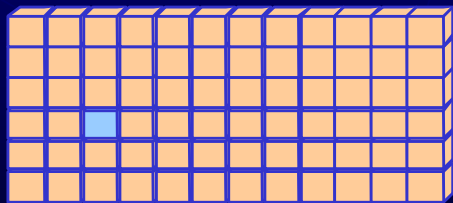
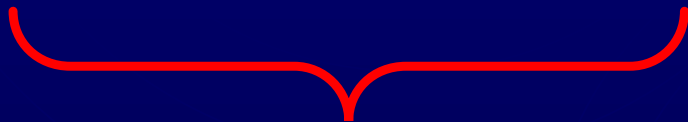
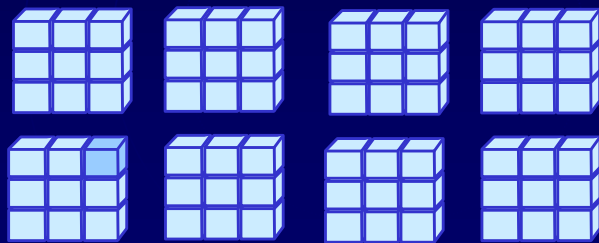
SHMEM

- Introduced on the Cray T3D
 - put, get, atomic swap, collectives
- Memory addressability
 - *symmetric objects*
 - stack, heap allocation on the T3D
 - Cray memory allocation routine *shmalloc*
- Characteristics
 - ordered in the original version on the T3D
 - out-of-order on the T3E due to adaptive routing
 - simple progress rules
 - simpler than MPI-2 1-sided, less synchronization
- Portable implementation from Ames - **GPSHMEM**



Global Arrays

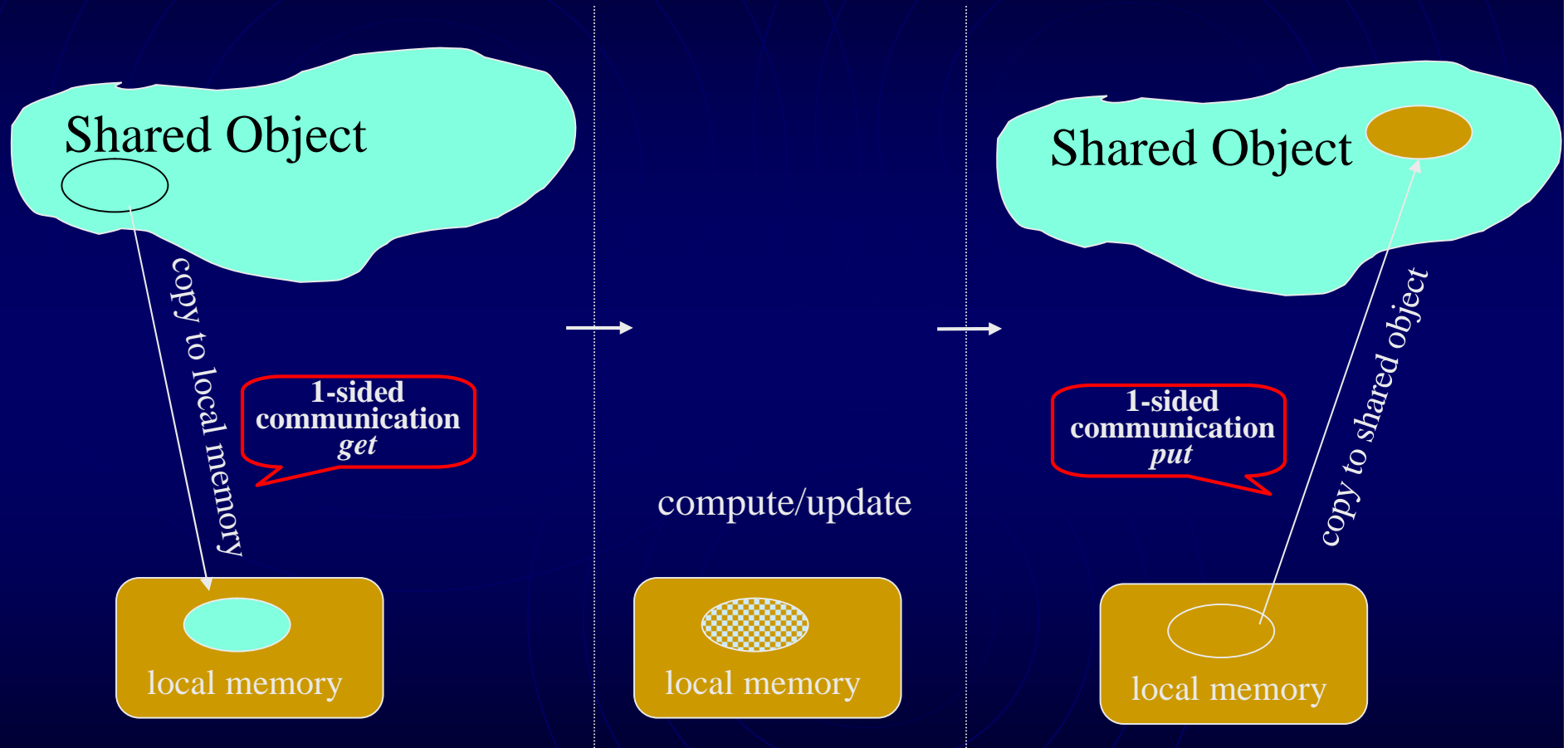
physically distributed data



- shared memory model in context of distributed arrays
- dual view of data
 - shared
 - distributed
- data locality control
- used in multiple areas
 - popular in parallel computational chemistry codes

single, shared data structure/ global indexing
e.g., $A(4,3)$ rather than $\text{buf}(7)$ on task 2

Global Array Model of Computations (motivated by NUMA hardware)

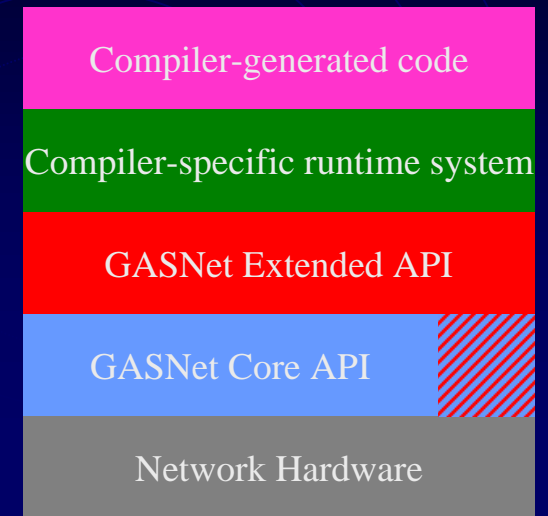


PModels Run-Time System Activities

- Communication support critical
 - Need performance, scalability, and portability
 - Low latency RMA communication
 - e.g., put, get, locks, atomic read-modify-write
 - nonblocking API for overlapping communication with data movement for latency hiding
 - both small message and bulk operations important
 - Collective operations
 - barriers, reduce, broadcast
- Run-time efforts represented by
 - GASNet (LBNL)
 - ARMCI (PNNL, OSU)

GASnet

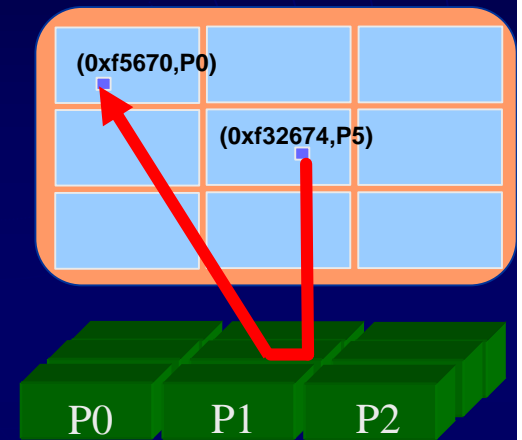
- LBNL system for UPC
- Aims for a wide portability
- 2-Level architecture to ease implementation
- Core API
 - Based heavily on Active Messages
 - Implemented directly on each platform
 - Most basic required primitives, as narrow and general as possible
- Extended API
 - Wider interface that includes more complicated operations
 - Reference implementation of extended API in terms of the core API
 - Implementors can choose to directly implement any subset for performance - leverage hardware support for higher-level operations



ARMCI

Aggregate Remote Memory Copy Interface

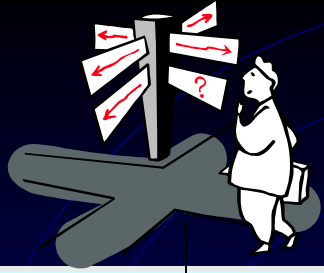
- Portable and low-level API used by
 - GPSHMEM, CoArray Fortran, Global Arrays
- Broad set of functionality
 - *put, get, accumulate* (also with noncontiguous interfaces)
 - *atomic read-modify-write, mutexes and locks*
 - collective operations
- Characteristics
 - simple progress rules, operations ordered w.r.t. target (ease of use)
 - does not assume any particular implementation model (e.g., AM)
 - focus on block data rather than single word transfers
- High performance delivered on a wide range of platforms
 - Multi-protocol and multi-method implementations



direct remote memory access

Integrated Run-Time System

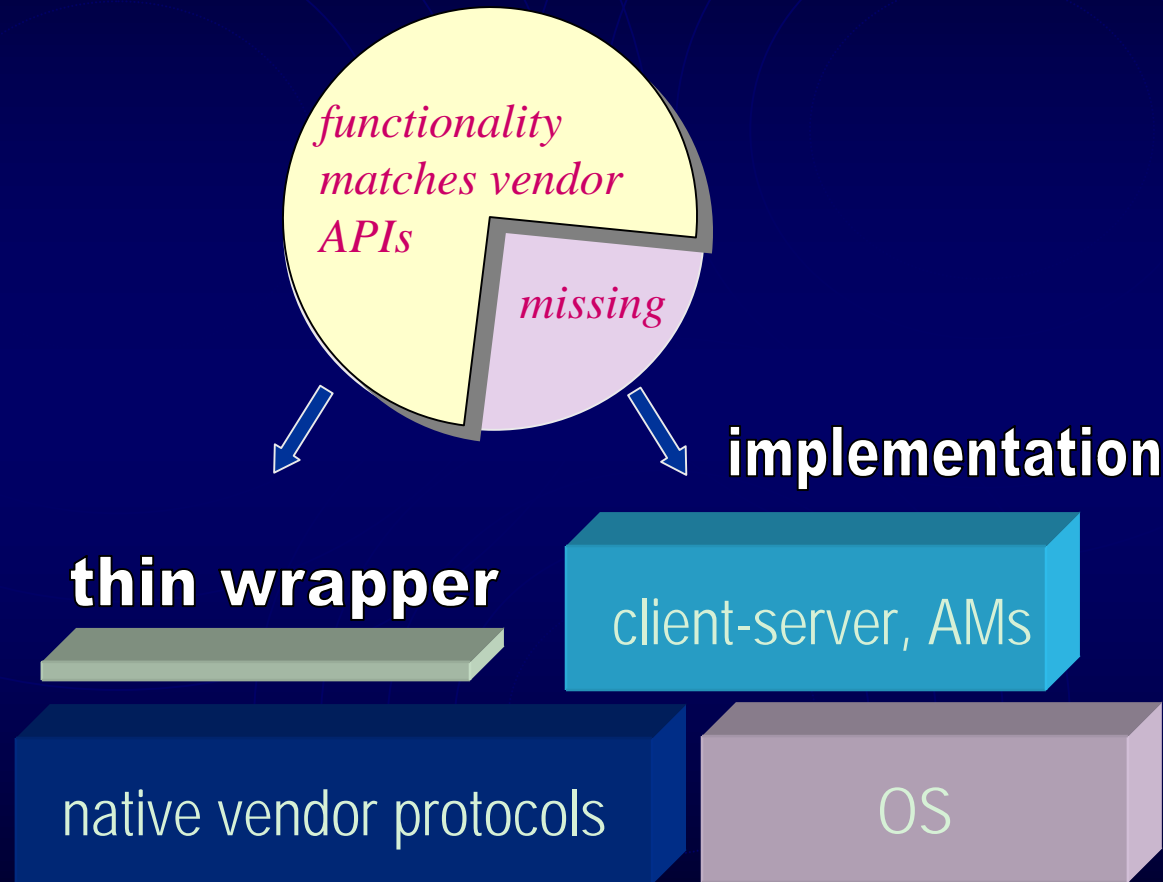
- Goal to build run-time system used accross PModels project
 - generality and flexibility
 - portability and performance
- Combine best features of ARMCI and GasNet, for example
 - AM-MPI developed by GasNet should work with ARMCI thus providing the Active Message API (polling only)
 - define extended set of memcopy interfaces in UPC able to use highly tuned ARMCI strided and vector interfaces
 - add nonblocking APIs defined by GasNet to ARMCI



Portability Challenge

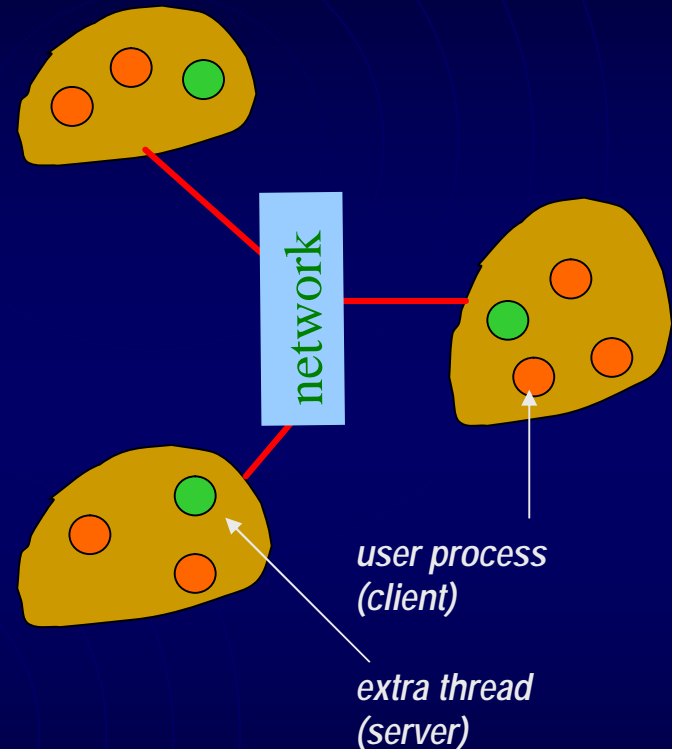
	put	get	accumulate	locks	RMW	ncontig API	nblock API	ordering	“regular” memory	extras
Cray SHMEM	Y	Y	N	N*	Ltd	N*	N	N	Y	
IBM LAPI	Y	Y	N	N	Y	Y*	Y	N	Y	thread safe, AM
Hitachi RDMA	Y	Y	N	N	Ltd	Ltd	Y*	Y	N	
Fujitsu MPlib	Y	Y	N	Y	N	Y	Y	Y	Y	
Myricom GM	Y	N	N	N	N	N	Y	Y	N	thread safe
VIA	Y	Opt	N	N	N	Y	Y	Y	N	thread safe
Quadrics	Y	Y	N	Y	Ltd	N	Y	N	Y	programmable NIC
TCP/IP	N	N	N	N	N	Y	Y	Y	Y	thread safe

Achieving Portability



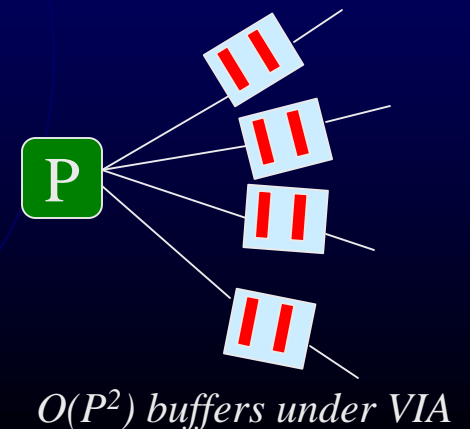
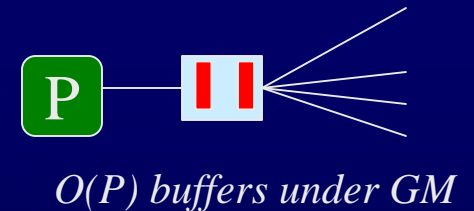
Client-Server Approach

- Used to implement missing functionality
 - conversion of memory requirements
 - e.g. dynamic memory registration
 - *get* on top of *put*
- Architecture
 - Extra thread on each SMP node used as server
 - Blocked when no requests available
 - Polling available as a configuration option if extra CPU power available
 - Interrupt/demand driven operation
 - usually adds extra cost
 - AM handler like approach but no full AM API



Scalability and Performance of RMA

- In principle RMA model is highly scalable
 - given good h/w support e.g., Cray T3E
 - otherwise might be limited by a s/w implementation
 - conversion of h/w, OS, and programming model requirements
- Depend on the underlying network capabilities
 - reliability (user or h/w/firmware?)
 - APIs and protocols - native RMA or other paradigm?
 - memory access requirements
 - registering might be required
 - flow control & buffer management (e.g., VIA vs GM)
- We want the network h/w handle as much as possible
 - Design of complex NIC can be costly!
 - Traditionally IBM SP designs did not favor low latency RMA



Relation To Blue Gene/L

- Interested in porting GAS models to the machine
- Advantages of GAS Models
 - complementary approach to MPI
 - no embedded synchronization
 - better suited for irregular applications and data access patterns
 - scalable implementation could be simpler
 - no message queues management, matching sends with receives
 - movement of data between two memory locations
 - shared memory style access to data
- Challenges
 - dealing with h/w faults - none of the GAS models supports it
 - underlying network protocols should be RMA aware for best performance
 - building RMA on top of message passing compromises performance

Final Thoughts

- GAS Models could offer an alternative to MPI on BG/L
 - we don't know which model would work best
 - implementation might be a limiting factor
 - good run-time support critical for performance and scalability
 - opportunities for using second processor
- Need to engage the IBM team to derive an **efficient** implementation of the PModels run time
 - network protocols and OS issues
 - performance characteristics and resource utilization
- Some research problems are difficult and span multiple s/w layers e.g., fault tolerance
 - programming model vs application vs OS support
 - as a minimum run-time system must recognize faults and provide data up to programming model